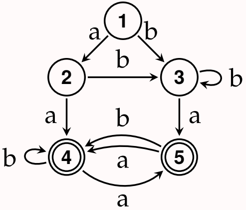
Fast Coalgebraic Bisimilarity Minimization

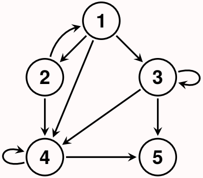
Jules Jacobs
Radboud University

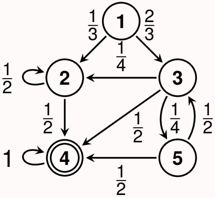
Thorsten Wißmann
Radboud University

The Automata Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...

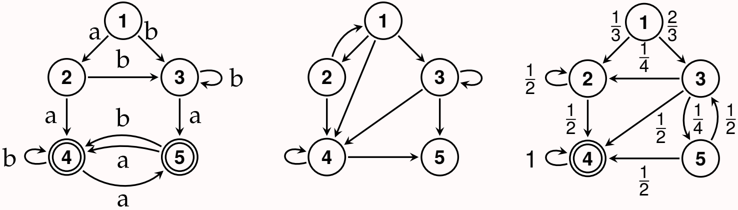






The Automata Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...

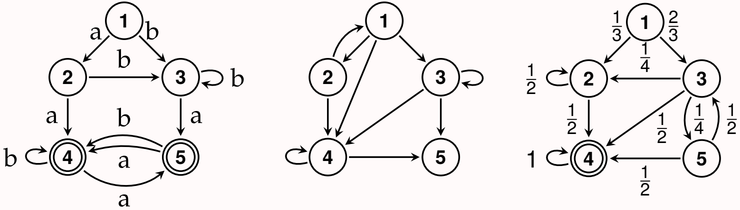


F-coalgebra

a unifying theory of automata and strong bisimilarity

The Automata Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...



F-coalgebra

a unifying theory of automata and strong bisimilarity

This Work

a **fast** and **general** algorithm for minimizing automata

What's an F-coalgebra?

What's an F-coalgebra?

Definition

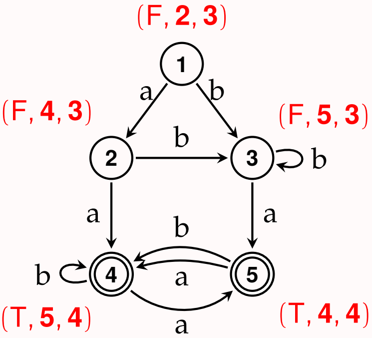
Let $F : \mathcal{C} \longrightarrow \mathcal{C}$ be an endofunctor on a category $\mathcal{C}$. An F**-coalgebra** is an object A of $\mathcal{C}$ together with a morphism $\alpha : A \longrightarrow FA$ of $\mathcal{C}$, usually written as (A, α). An F-coalgebra homomorphism from (A, α) to another F-coalgebra (B, β) is a morphism $f : A \longrightarrow B$ in $\mathcal{C}$ such that $Ff \circ \alpha = \beta \circ f$. Thus the F-coalgebras for a given functor F constitute a category.

What's an F-coalgebra?

Definition

Let $F : \mathcal{C} \longrightarrow \mathcal{C}$ be an endofunctor on a category $\mathcal{C}$. An F**-coalgebra** is an object A of $\mathcal{C}$ together with a morphism $\alpha : A \longrightarrow FA$ of $\mathcal{C}$, usually written as (A, α). An F-coalgebra homomorphism from (A, α) to another F-coalgebra (B, β) is a morphism $f : A \longrightarrow B$ in $\mathcal{C}$ such that $Ff \circ \alpha = \beta \circ f$. Thus the F-coalgebras for a given functor F constitute a category.

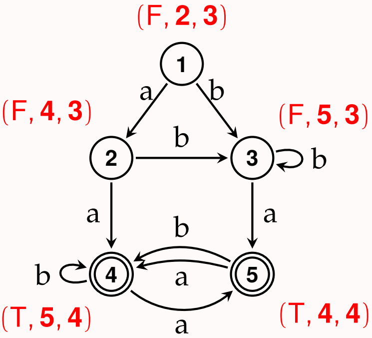
Finite coalgebras unify automata

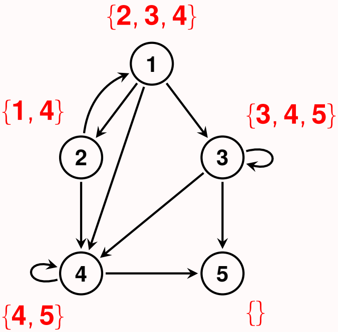


DFA

$C \to \{F, T\} \times C \times C$

Finite coalgebras unify automata





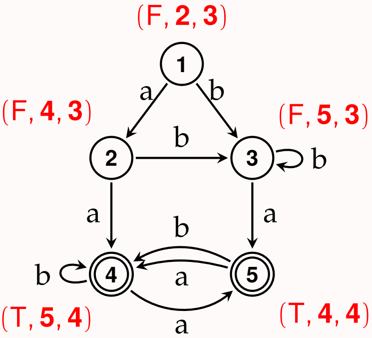
DFA

$C \to \{\mathsf{F}, \mathsf{T}\} \times C \times C$

Transition system

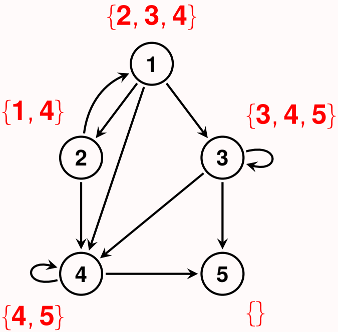
$C \to \mathcal{P}(C)$

Finite coalgebras unify automata



DFA

$C \to \{\mathsf{F}, \mathsf{T}\} \times C \times C$



Transition system

$C \to \mathcal{P}(C)$

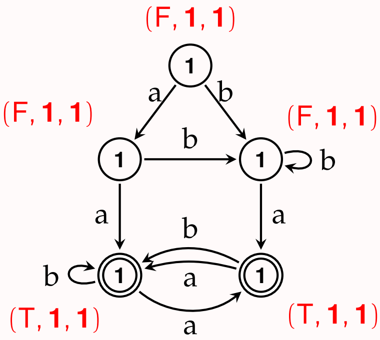
Labeled transition system

$C \to \mathcal{P}(A \times C)$

Markov Decision Process

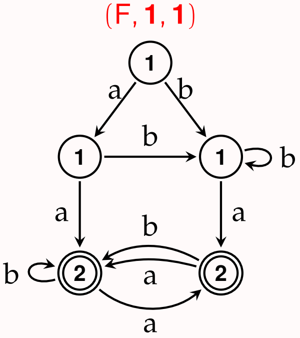
$C \to \mathcal{P}(\mathcal{D}(C))$

Minimizing a DFA



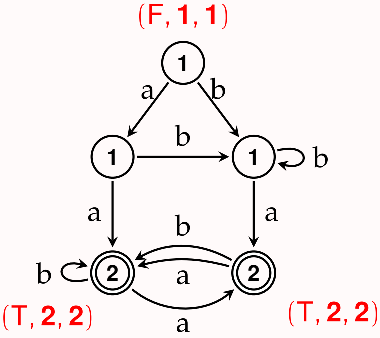
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a DFA



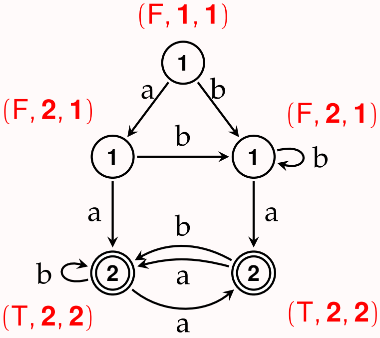
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a DFA



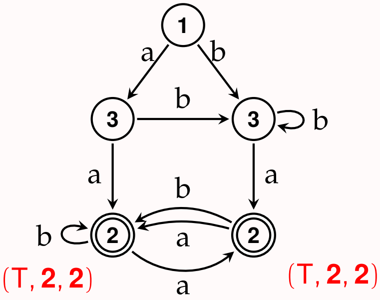
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a DFA



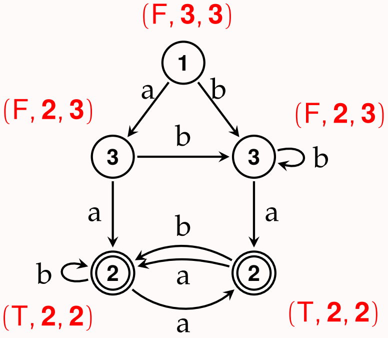
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a DFA



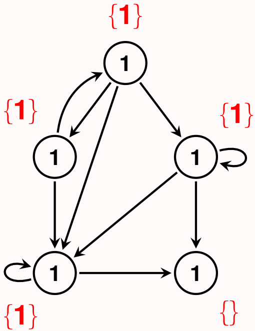
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a DFA



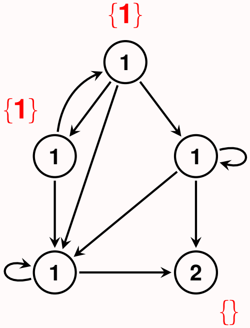
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a transition system



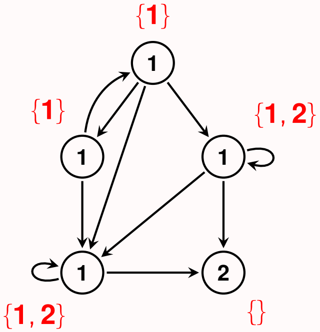
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a transition system



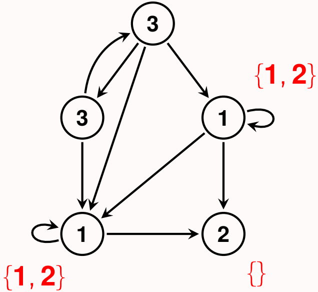
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a transition system



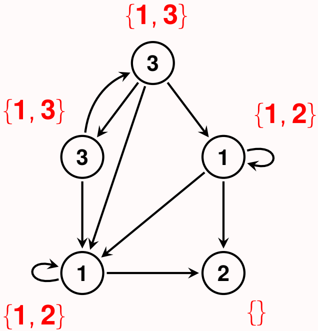
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a transition system



- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Minimizing a transition system



- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block
- How many times does a signature get computed?

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block
- How many times does a signature get computed?
  - at most $O(\log n)$ times per edge

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block
- How many times does a signature get computed?
  - at most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block
- How many times does a signature get computed?
  - at most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations
  - Total complexity: usually $O(km \log n)$

What we need from the automaton

- Ability to compute signatures
- Ability to determine predecessors

Complexity

- How many times does a state's number change?
  - at most $O(\log n)$ times, if we use the old state number for largest new block
- How many times does a signature get computed?
  - at most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations
  - Total complexity: usually $O(km \log n)$
- What about the complexity of bookkeeping?
  - See paper for n-way partition refinement data structure

Comparison

Boa Our algorithm

Comparison

Boa Our algorithm

CoPaR Asymptotically more efficient
($O(m \log n)$ signatures vs $O(m \log n)$)
Applicable to zippable functors

Comparison

Boa Our algorithm

CoPaR Asymptotically more efficient
($O(m \log n)$ signatures vs $O(m \log n)$)
Applicable to zippable functors

DCPR Distributed coalgebraic algorithm
Same generality as us
Quadratic complexity

Comparison

Boa Our algorithm

CoPaR Asymptotically more efficient
($O(m \log n)$ signatures vs $O(m \log n)$)
Applicable to zippable functors

DCPR Distributed coalgebraic algorithm
Same generality as us
Quadratic complexity

mCRL2 Specialized C++ algorithm suite
for labeled transition systems

$$C \to \mathcal{P}(A \times C)$$

Several LTS minimization algorithms
from literature ('90, '03, '17, '19)

benchmark		**time (s)**			**memory (MB)**	
type	**n**	*CoPaR*	*DCPR*	*Boa*	*DCPR*	*Boa*
fms	**1639440**	232	84	1.12	514×32	196
	4459455	–	406	4.47	1690×32	582
wlan	**607727**	105	855	0.28	147×32	42
	1632799	–	2960	0.79	379×32	93
wta(W)	**152107**	566	79	0.74	642×32	83
	944250	–	675	11.96	6786×32	1228
wta(Z)	**156913**	438	82	0.48	677×32	92
	1007990	–	645	16.75	5644×32	1325
wta(2)	**154863**	449	160	0.81	621×32	79
	1300000	–	1377	23.35	7092×32	1647

What is the cost of generality?

benchmark		time (s)		memory (MB)	
type	**n**	*mCRL2*	*Boa*	*mCRL2*	*Boa*
cwi	**2416632**	13.9	1.4	1780	249
	7838608	214.2	15.8	5777	814
	33949609	282.2	31.5	16615	2776
vasy	**6020550**	33.8	3.1	2124	520
	11026932	51.6	6.1	2768	619
	12323703	56.9	7.0	3103	734

For *mCRL2*, we pick its best algorithm and its self-reported time.
For *Boa*, we report wall-clock time.